Carnegie Mellon University
Software Engineering Institute

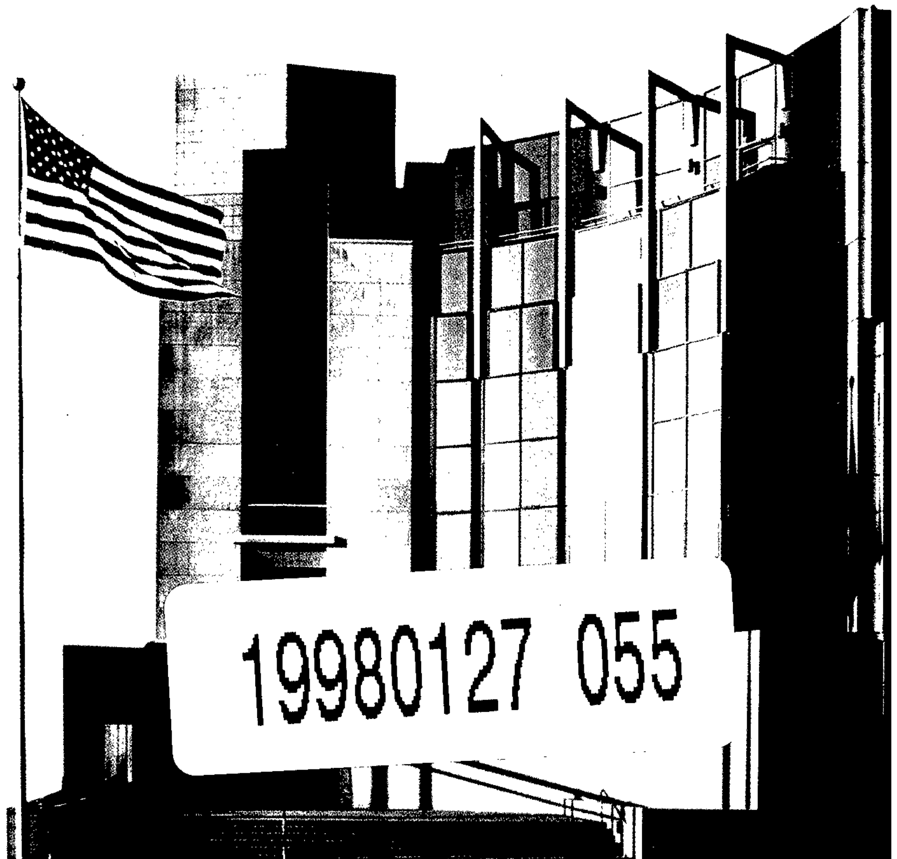
A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain

Andreas Polze
Daniel Plakosh
Kurt C. Wallnau

December 1997

TR

TECHNICAL REPORT
CMU/SEI-97-TR-011
ESC-TR-97-011



Technical Report
CMU/SEI-97-TR-011
ESC-TR-97-011
December 1997

A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain

Andreas Polze (Humboldt University of Berlin)
Daniel Plakosh
Kurt C. Wallnau

Dynamic Systems

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Table of Contents

List of Figures

Acknowledgements

The work described in this report was funded by the NIST Manufacturing Engineering Laboratory (MEL). Additional resources were provided by Humboldt University of Berlin and the Software Engineering Institute (SEI). Thanks to Neil Christopher of NIST/MEL for making all the right technical connections between SEI investigations of distributed object technology and the manufacturing domain. Also thanks to Evan Wallace and Fred Proctor (both from NIST/MEL) for providing detailed technical assistance—and code—for the motion controller.

A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain

Abstract: The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is an important and popular technology that supports the development of object-based, distributed applications. The benefits promised by CORBA (abstraction, heterogeneity, etc.) are appealing in many application domains, including those that satisfy real-time requirements—such as manufacturing. Unfortunately, CORBA was not specified in light of real-time requirements, and so the question remains whether existing object request brokers (ORBs) can be used in real-time settings, or whether developers of real-time systems must await future extensions of CORBA that address real-time issues or use non-CORBA-compliant ORBs. In this report, we describe the application of an off-the-shelf ORB to two real-time model problems. Based on our experiences, we believe that today's ORBs can be used in real-time settings, with certain caveats as outlined in this report. We also outline the concept of *composite objects*, an approach for extending the range of non-real-time ORBs into a greater variety of real-time settings.

1 Introduction

Real-time computing is often associated with special purpose systems which are vendor specific, expensive, hard to maintain, and difficult to upgrade. Examples include embedded computer systems in today's airplanes, cars, and automated factories. As computers become increasingly prevalent in society, and as computing applications become increasingly complex, so, too, will there be an increasing demand to integrate multiple real-time control systems into larger systems, and to integrate real-time subsystems into non-real-time distributed computing environments. This way, non-real-time components like graphical user interfaces and databases can be connected to a system with real-time components.

Distributed computing using commercial off-the-shelf (COTS) hardware and software components is appealing because of its cost effectiveness. A number of standards have evolved over the past few years which ensure interoperability among heterogeneous hardware platforms and software packages from different vendors. Object technology has been used to describe interfaces and interaction patterns for such distributed applications. The Common Object Request Broker Architecture (CORBA) [OMG 95, Soley 95] is the most successful representative of an object-based distributed computing infrastructure. There are a number of commercially available implementations of CORBA, which we refer to as object request brokers (ORBs[1]) as a shorthand.

1. Throughout this report the term "ORB" will be used to refer to existing implementations of CORBA.

In this paper, we investigate the use of CORBA in the real-time computing domain. This question is interesting and important because CORBA was not designed originally for real-time applications, and yet there is evident demand for applying CORBA in real-time settings. We view this question as a form of technology evaluation, where a technology (in this case, CORBA) is evaluated with respect to concrete problems within an application domain (in this case, shop-floor automation). One technique that is particularly useful in technology evaluation is the use of model problems—focused experimental prototypes that reveal technology benefits and limitations in well-bounded ways. (For a more general overview of technology evaluation techniques, see "A Framework for Evaluating Software Technology" [Brown 96].) The Software Engineering Institute (SEI) Simplex and the National Institute of Standards and Technology (NIST) Real-Time Control System (RCS) architectures provide the basis for two model problems described in this report.

We use the Simplex architecture [Sha 96] as the basis for a model problem that allows us to explore the use of ORBs as an interconnection mechanism *between* hard real-time[2] systems. The Simplex application provides coordinated control over two inverted pendulums (each of which is a Simplex-based, fault-tolerant, hard real-time control system). We use the NIST RCS architecture as the basis for a second model problem that allows us to explore the use of available ORBs as a real-time communication mechanism *within* a real-time system. The RCS application is the NIST Motion Controller, a simulation of a device that receives numerical control (NC) instructions and mills an appropriate surface.

The rest of this paper is organized as follows: In Section 2, we summarize the key background concepts of the Simplex and RCS architectures; we also briefly describe CORBA and two top-level approaches for integrating CORBA with real-time systems. In Section 3 we discuss the coordinated inverted pendulum model problem, while in Section 4 we discuss the NIST motion controller model problem. In Section 5, we outline the key concepts underlying composite objects, an approach for predictable integration of CORBA with real-time software. In Section 6 we provide an outline of related work. Finally, we present our conclusions in Section 7.

2. By "hard real-time" we mean systems that must be guaranteed to meet execution schedules. In contrast, "soft real-time" systems might provide only probabilistic guarantees (for example, meet execution schedules 90% of the time).

2 Background

2.1 CORBA

CORBA is a specification for object-based interprocess communication in distributed, heterogeneous environments. CORBA has been standardized by the Object Management Group (OMG), an industry group of over 600 computer manufacturers and independent software vendors. The current version of CORBA, version 2.0 [Soley 95], was specified in 1995. There are many COTS implementations of CORBA available in the marketplace; we refer to such implementations as object request brokers (ORBs), and are careful to distinguish between CORBA (a specification) and ORB (an implementation).

CORBA provides transparent distribution of objects and supports interoperability across several dimensions of heterogeneity. Components of a CORBA program can be implemented in different programming languages on a variety of operating systems and hardware platforms. CORBA enables client programs to invoke a server object's methods regardless of whether the server object is in the client's address space or located on a remote node in a distributed system. CORBA defines services that locate a server object implementation, prepare the server object for receiving a client's request, and finally communicate data making up a request.

CORBA Interface Definition Language (IDL) is a declarative language that describes the interfaces to server object implementations, including the signatures of all server object methods that are callable by clients. The surface syntax of IDL is similar to that of C++, but IDL is language-independent. Mappings from IDL to C, C++, Ada, and Smalltalk have been specified. However, IDL does not provide syntax for object implementations.

The OMG sponsors a number of technical initiatives aimed at fostering the adoption of CORBA. One ongoing effort within the OMG that is of relevance to the work described in this report concerns real-time CORBA (RT-CORBA). An RT-CORBA special interest group (RTSIG) is investigating possible future extensions to the OMG Object Management Architecture (OMA)[3] to support a wide spectrum of distributed, real-time, fault-tolerant systems. It must be noted that neither CORBA nor the OMA currently address questions of real-time quality of service; in the remainder of this report, the terms CORBA and ORB will be used to refer to existing, non-real-time specifications and products.

The focus of the work described in this report is on the use of ORBs available today in real-time and fault-tolerant settings. This focus provides risk reduction against the possibility that the OMG RTSIG will fail to achieve industry consensus on proposed real-time extensions to

3. The OMA consists of CORBA plus a collection of object services hosted on CORBA. These object services support general programming constructs (such as naming services and event channels) as well as vertical industry-specific services (such as work flow, telecommunications, and manufacturing).

CORBA and the OMA. Additionally, the work provides a useful transition step even if the RT-SIG is successful, because even in this scenario, the marketplace will require some (perhaps significant) time to produce robust implementations of any real-time extensions to CORBA and the OMA.

Also, it should be borne in mind that all such extensions will ultimately rely on operating systems services that provide the necessary real-time quality-of-service guarantees; for the foreseeable future, such operating systems will continue to represent (relatively speaking) a niche market. Furthermore, most real-time operating systems currently fail to support real-time communication over a network—and this is obviously relevant to RT-CORBA. Therefore, the work described in this report will remain relevant even with the advent of time and industry acceptance of RTSIG proposals.

This report addresses two different scenarios for integrating CORBA with real-time applications:

- Use CORBA as a gateway to real-time subsystem(s). In this scenario, the CORBA gateway is not responsible for satisfying real-time, quality-of-service guarantees.
- Use CORBA as a communication mechanism within a real-time subsystem. In this scenario, the ORB must contribute to (or not hinder) real-time quality of service; in particular, it must satisfy potentially stringent performance requirements.[4]

To explore the first scenario, we extended an existing model problem developed previously on the Simplex architecture (described in detail, below). For the second scenario, we extended an existing system developed by the NIST Manufacturing Engineering Laboratory (MEL) that was developed on the NIST RCS architecture.

2.2 The Simplex Architecture

Current real-time computer software architectures do not support the safe and reliable insertion of new components and technologies into existing systems. The Simplex architecture [Sha 96] has been developed to support the reliable *online* upgrade of hardware and software components; reliability is maintained in spite of errors in new modules. This is important when introducing changes, such as new COTS components, into running systems.

To mitigate the risks of a system upgrade, the Simplex architecture has been designed to tolerate timing faults such as overrun, programming system faults such as illegal addressing, and semantic faults due to modeling, algorithm design, or implementation errors. Simplex handles timing faults and programming system faults by temporal and spatial encapsulations. Software semantic faults are handled by the use of analytic redundancy [Sha 96]. The Simplex architecture is based on generalized rate-monotonic scheduling theory [Rajkumar 94]. Also, the

4. As discussed later, priority inversion is another area where embedded ORBs can interfere with satisfying real-time requirements. However in the motion controller model problem, this was not an issue.

Simplex architecture assumes that the underlying operating system supports either the priority inheritance protocol or the priority ceiling protocol to avoid priority inversion problems during the management of shared resources. The current implementation of Simplex is based on interprocess communication mechanisms specific to an operating system compliant with POSIX 1003.1b.

The Simplex architecture has been used successfully to solve a number of control problems, and a number of demonstration applications have been implemented using Simplex. These solutions demonstrate how real-time software can be altered and tested without making the controlled device unsafe. A number of controllers can be interchanged, for example, to implement new or improved control algorithms. In case of faults introduced by the new controller, the system implements graceful degradation, ultimately relying on a safety controller that implements fail-safe operational behavior.

The basic building block in the Simplex architecture is the *replacement unit*, containing one or more processes with a communication template that facilitates the online replacement of one unit with another. Replacement units are designed in such a way that they can be added, deleted, merged, or split online by a set of standardized upgrade transactions.

In the Simplex architecture, autonomous subsystems like those found in typical industrial systems are implemented as application modules. The application module implements software fault-tolerance mechanisms and acts as a software fault-containment unit, with its components running in their own address spaces. As illustrated in Figure 2-1, an application module is composed of specialized replacement units: application units (controllers), a module management unit (decision module), and a safety unit:

- The application units implement the application-specific functionality of a given subsystem.
- The module management unit determines when an application unit is leaving its designated safety region and switches control to an alternative application unit (and possibly to the safety unit, the fail-safe application unit).
- The safety unit implements a safety controller when the physical subsystem is not fail-safe.

Communication between the components of an application module is realized via a real-time publish/subscribe model. The current implementation, called *tagged data interprocess communication*, is based on POSIX message queues as available in LynxOS. The operating system's fixed-priority scheduling policy is used to give the decision module and device I/O manager highest priority in the system. Input from the device is distributed to all the replacement units acting as controllers. Output from the controllers is monitored by the decision module. If output from one of the application-specific controllers is missing or leaves the range of valid output data (as provided by the safety controller), the particular controller is assumed to be faulty and is no longer used.

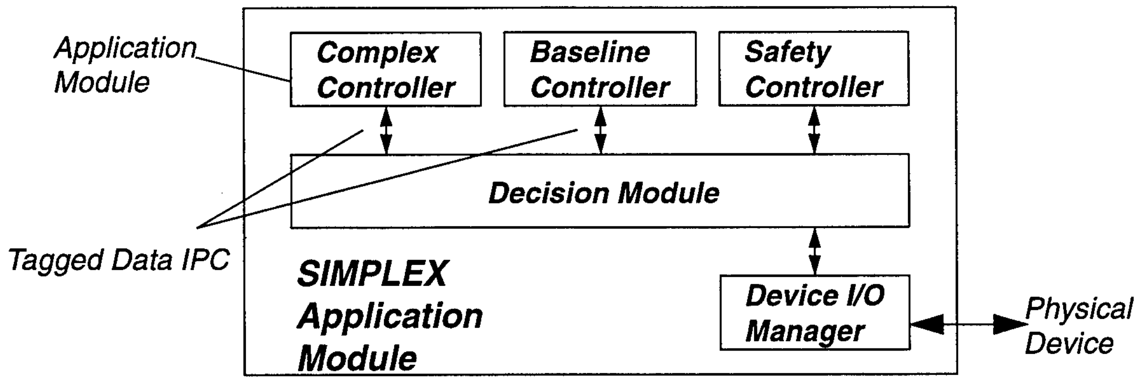


Figure 2-1: Simplex Architecture

2.3 The Real-Time Control System (RCS) Architecture

The NIST RCS architecture is a reference architecture for manufacturing applications. The RCS architecture abstracts implementation-dependent details of real-time control software as well as underlying computing platforms. RCS provides to the application developer a unified interface to operating systems and hardware devices. Code using this interface can be ported between different platforms and applications.

The RCS library (RCSlib) [NIST URL] is a C++ class library to support the development of multi-platform real-time distributed applications. The RCSlib provides synchronization and communication constructs, functions for communicating data between controller components, and RCS-specific functions for sensory processing, task decomposition, world modeling, and the operator interface. The RCSlib has been ported to several hardware and operating system platforms.

The Communication Management System (CMS) provides access to a fixed-size buffer of general data to multiple reader or writer processes on the same processor, across a backplane, or over a network. Regardless of the communication method required, the interface to CMS is uniform. Methods are provided to encode all of the basic C data types in a machine-independent format, and to return them to the native format. The Neutral Manufacturing Language (NML) provides a higher level interface to CMS. NML provides a mechanism for handling multiple types of messages in the same buffer as well as simplifying the interface for encoding and decoding buffers in neutral format and the configuration mechanism.

Figure 2-2 illustrates the structure of a typical RCS application using NML. The application is distributed across three computers. Processes 1, 2, and 3 are able to write directly into the shared-memory buffers they use because they are all located in the same computer or backplane. Processes 4, 5, and 6 can only access the buffers through an NML Server (referred to as *remote* processes in RCS).

NML servers must be run for each buffer that will be accessed by remote processes. They read and write to the buffer in the same way as local processes on the behalf of remote processes. NML is message based rather than stream based.

We investigate the effect of replacing messages from NML by CORBA remote-method invocations. Using CORBA inside a real-time application would offer the advantages of object technology—like encapsulation, data abstraction, and structuring—to the real-time programmer.

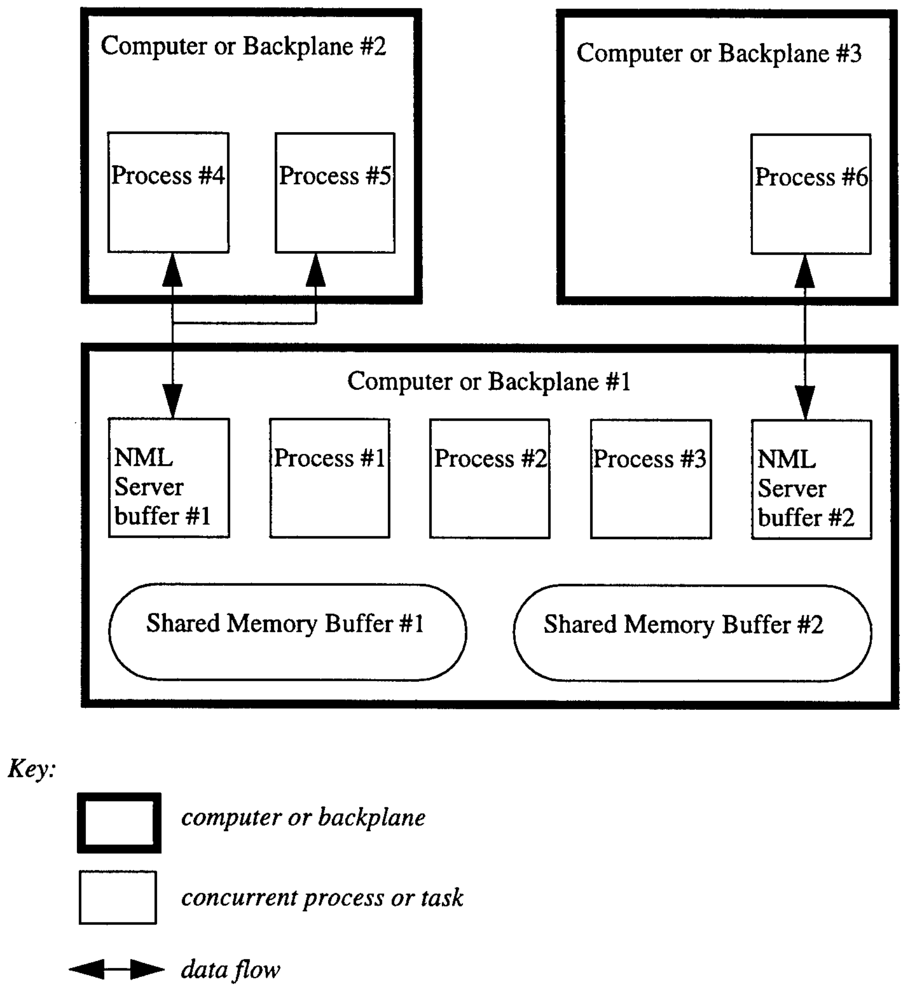


Figure 2-2: Structure of an Application Using RCSlib

3 The Synchronized Inverted Pendulum

We used the synchronized inverted pendulum as a model problem to explore the use of CORBA as a non-real-time gateway between real-time subsystems. By extending a real-time system like Simplex with a CORBA gateway interface, a real-time application can interact with other components in a distributed environment. For instance, an administration and monitoring component could be connected to the real-time application. The CORBA gateway appears to the system as a different kind of real-time process. It is not assumed to provide output to the controlled physical device; however, it may communicate with the application-specific controllers to retrieve state informations and modify system variables. Thus, a second, outer control loop can be established among multiple real-time applications through the CORBA gateways (whereas the inner control loops are hard real-time control subsystems).

To implement the model solution, we ported the Xerox inter-language unification (ILU) ORB (version 2.0alpha9) to the LynxOS v2.4 real-time operating system. While ILU is not fully complaint with CORBA v2.0, it provides sufficient capabilities and is based upon technical principles with sufficiently close proximity to CORBA, to represent a reasonable exemplar of CORBA technology. The decision to port ILU was one of expediency: no *commercial* ORBs were (yet) available on LynxOS, the real-time operating system that supports Simplex.

3.1 Model Problem Description

The *inverted pendulum* is a Simplex demonstration application, where a physical device is controlled through fault-tolerant software. The rod of the inverted pendulum is attached to a cart by a freely swinging but unpowered hinge. The cart can move horizontally under computer control. The pendulum is instrumented to provide the rod angle and cart position on the track as signals to the controlling computer. Three different controllers in the Simplex system implement more or less sophisticated algorithms for keeping the rod upright. Every controller tries to position the cart close to a given point on the track (the set point). By changing the value for the set point, the cart can be moved along the track. However, keeping the pendulum balanced has a higher priority than approaching the position given by the set point.[5]

Figure 3-1 illustrates the more complex *coordinated* pendulum scenario, where two Simplex inverted pendulum applications are controlled via an application that executes control via CORBA communication links. Both Simplex application modules are extended by a CORBA gateway object. The X-window-based graphical user interface acts as a CORBA client and polls position data (track position, angle of the rod) from the two Simplex applications. Furthermore, in response to user actions, the graphical user interface (GUI) component calls a method at the Simplex gateway objects to change the value of the set point. Thus, the user can

5. This refers to priorities within the control algorithm. Client requests to change the set point have a still lower priority.

move interactively the inverted pendulums, simulating remote control (and coordination) of real-time devices from a non-real-time subsystem.

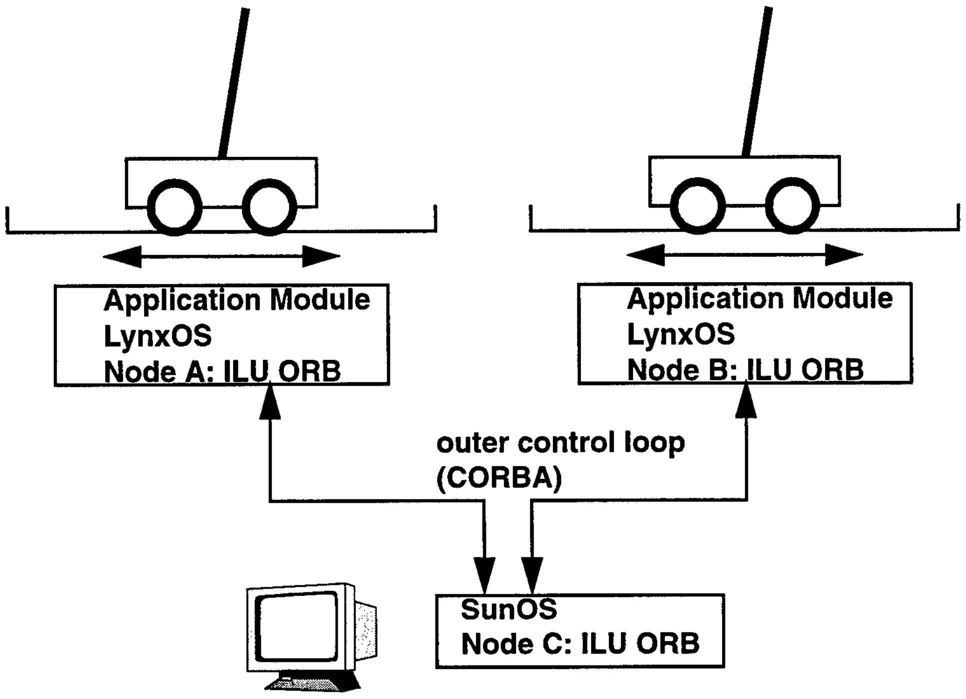


Figure 3-1: Synchronized Inverted Pendulum Scenario

Figure 3-2 shows the basic architecture of a Simplex-based application extended by a CORBA gateway. Two application-specific replacement units (complex and baseline controller) are shown as well as the safety controller and the device I/O manager.

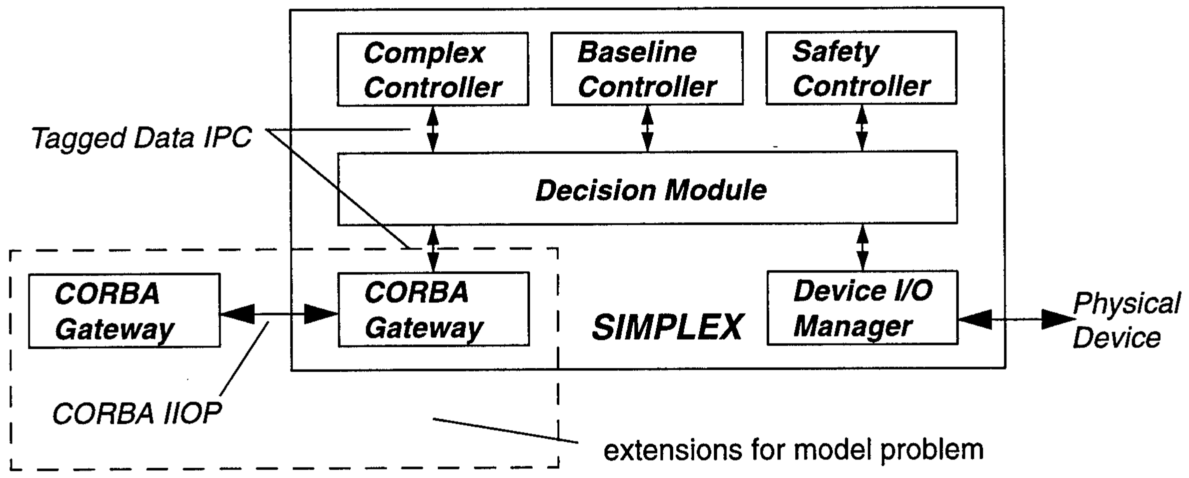


Figure 3-2: Extending Simplex with CORBA Gateways

Figure 3-3 shows the ILU Interface Specification Language (ISL) specifications for the methods implemented in the Simplex gateway objects. The ILU ISL is similar although not identical to CORBA IDL.

```
INTERFACE Pendulum;

CONSTANT serverID : ilu.CString = "Pendulum-Server" ;

TYPE T = OBJECT METHODS
      getPosition(
            OUT position:REAL,
            OUT angle:REAL,
            OUT refpoint:REAL),

      setReference(IN refpoint:REAL),

      startComputation()
END;
```

Figure 3-3: Interface Definitions for GUI/Pendulum Communication

Besides methods for getting pendulum status information and for changing the set point, Figure 3-3 contains a method `startComputation()`. The implementation of this method simulates a compute-intensive task and makes use of `malloc()` to stress the real-time system's resource management. This method permits us to simulate (and study the impact of) interference between the resource assumptions underlying both real-time and non-real-time applications that share common computing resources (as is implied by the CORBA gateway). Changing the frequency with which data is polled from the Simplex applications is another way to change the load on the real-time system—we varied this value between 10Hz and 100Hz.

Besides the Simplex gateway, we have introduced another CORBA object to the scenario shown, as illustrated in Figure 3-4.[6] This additional component performs trajectory planning and translates abrupt changes in the set point's value into a series of smaller changes which are performed with a frequency of 10Hz. Figure 3-4 also shows the default frequencies for the several communication steps in the synchronized pendulum scenario. As mentioned earlier, the graphical user interface allows user-initiated changes in the polling rate. Shared memory is used for interactions between the Simplex gateway and the other components of the lower-level Simplex system.

6. The implementation of this trajectory planning object was not completed. The discussion is included in the report as an indication of how additional server-side objects could be introduced into the model problem in order to extend the investigation of CORBA, possibly as part of a demonstration of composite objects.

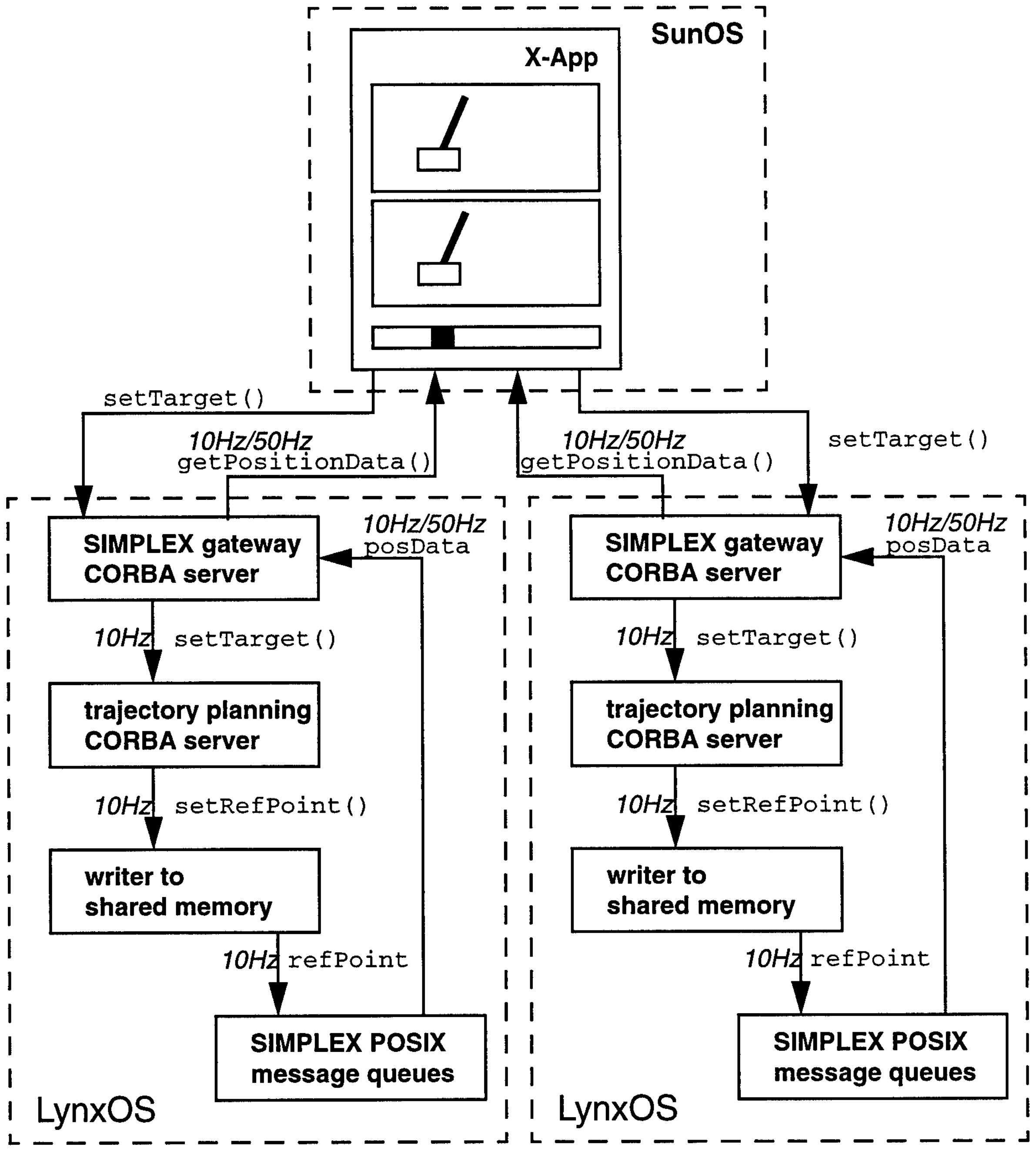


Figure 3-4: Communication Structure for the Synchronized Inverted Pendulum

We implemented CORBA communication using a multi-threaded version of ILU; in the Pendulum-gateway object, a separate thread per method and per client is created. All ILU-method-invocations are executed by low-priority threads on the LynxOS system. Other components of Simplex run as high-priority processes.

3.2 Observations

In our specific application scenario, ILU CORBA proved to be very usable to connect the graphical user interface to the synchronized pendulums. One important point is that the ORB objects sharing the real-time platform with Simplex did not interfere with Simplex because the object implementation executed at a lower priority than the Simplex tasks. Priority scheduling is a common and useful device in real-time application development, and provides one dimension of firewall between ORB and real-time applications. However, communication latency can still be a source of problems in coordinating two real-time applications across a non-real-time gateway.

Although measurements show a substantial variation of communication latency when using ILU, those variations are not immediately visible to the user. There are two reasons why CORBA is adequate despite these variations in latency. First, the Simplex control loop—although running with a 20 ms period—needs more than 100 ms to reestablish balance, once the pendulum has been disturbed. Since reestablishing balance has a higher priority than changing the pendulum set point (which disturbs the balance of the pendulum), Simplex always takes some time to react to changes to the cart's position—and the varying CORBA communication latency is absorbed by this time. Second, it is hardly noticeable for the user whether updates to the pendulum's image on the screen occur exactly every 100 ms or if there are small variations in screen update. Therefore, we draw the conclusion that today's CORBA can be used to provide soft real-time control and monitoring of hard real-time devices.

However, although occasional execution of CORBA methods on the computer platform shared with the real-time application worked well, we also demonstrated that unconstrained execution of CORBA methods may disturb the real-time system's behavior. CORBA does not explicitly support concepts such as call admission. Therefore, by just invoking enough methods, the ORB on the real-time system may put unacceptable load onto the system. Indeed, we could show that with a sufficiently high number of pendulum GUIs connected to the same Simplex application, the networking load on the LynxOS system increases to the extent that real-time tasks get delayed; deadlines are missed; and the pendulum finally falls down.[7]

Therefore, some form of *firewall* between CORBA and the real-time application needs to be constructed so that CORBA can not interfere with the assumptions of the real-time application. For this purpose, we propose the concept of *composite objects,* which is outlined briefly in Section 5.

[7]. In this case, we exploited a priority inversion problem in the LynxOS whereby interprocess communication (IPC) requests by tasks with a low priority cause network interrupts to interfere with the higher-priority control algorithms. Some form of client-side call admission would have resolved this, perhaps as provided by composite objects. The new version of the LynxOS also solves this problem directly at the network level by introducing priority scheduling.

4 The NIST Motion Controller

A real-time system could be restructured to exploit CORBA fully. Because of CORBA's object-oriented nature, this approach offers all the advantages of object technology to the developer of a real-time system. However, it is not clear whether today's ORBs can provide a sound basis for the real-time communication behavior required in this case. We study experimentally the effect of using CORBA communication as a replacement for shared-memory communication inside the NIST motion controller soft real-time application.

4.1 Model Problem Description

The motion controller is an application developed by NIST/MEL that uses RCS to control a machine tool; currently, the motion controller operates as a simulation. The machine tool has three axes; therefore all components deal with 3-tupels to specify tool motion. The application consists of three interacting components:

- `emove`: a component that translates RS274 CNC to "move" commands with velocity and acceleration information that are then forwarded to `traj`
- `traj`: a trajectory generator which performs interpolation of `emove` commands into finer grained moves
- `servo`: a simulated electro motor which represents the actual machine tool

The `traj` and `servo` components are implemented as periodic tasks. `servo` runs with a period of 10 ms (while the period for `traj` is 100 ms) and reads fresh trajectory information every tenth cycle. In the initial version of the application, both components accessed the same shared-memory locations. The RCSlib implementation provides an independent clock to each client process. Since `servo` and `traj` are not explicitly synchronized, `servo` may occasionally get out of synchronization with `traj`. This results in abrupt changes in the simulated motor's behavior. `servo` records the motor control data which can be plotted—jumps in the plotted curves indicate the impact of changes in the latency (i.e., "jitter") of communication between `traj` and `servo`.

We replaced the motion controller's shared-memory communication with communication through CORBA shared-memory objects (as with the Simplex model problem we used Xerox ILU). This way we gain location transparency of all the components; however, we have to deal with CORBA's communication "jitter," i.e., variations in communication latency. The motion controller model problem allows us to explore the impact of this variation.

Figure 4-1 shows the communication structure of the CORBA-based motion controller. We have introduced `shmServ`, a CORBA object, which implements four shared-memory structures. Read and write methods allow access to those data structures. Figure 4-2 describes the interface to `shmServ` in the ILU ISL. As noted earlier, although the surface syntax of ISL varies from CORBA IDL, both convey similar information (and in fact cross compilers for ISL to IDL have been developed by Xerox).

Note that we have changed the semantics of the motion controller demo somewhat by just introducing `shmServ`; in contrast to the original implementation, where concurrent accesses to shared memory have been synchronized based on word entities, the CORBA `shmServ` synchronizes accesses on a per-method basis. Therefore, we have created a tighter coupling between `traj` and `servo`.

However, this tighter coupling—resolving concurrent accesses based on the application-specific data structures, seems to be more correct than the initial approach. With the shared-memory-based implementation of the motion controller demo, when trying to get a 3-tupel describing the tool's next position, `servo` might read one old and two new data components—which certainly is incorrect. To evaluate the effect of the extra synchronization introduced by `shmServ`, we have implemented that component in a single-threaded and a multi-threaded version.

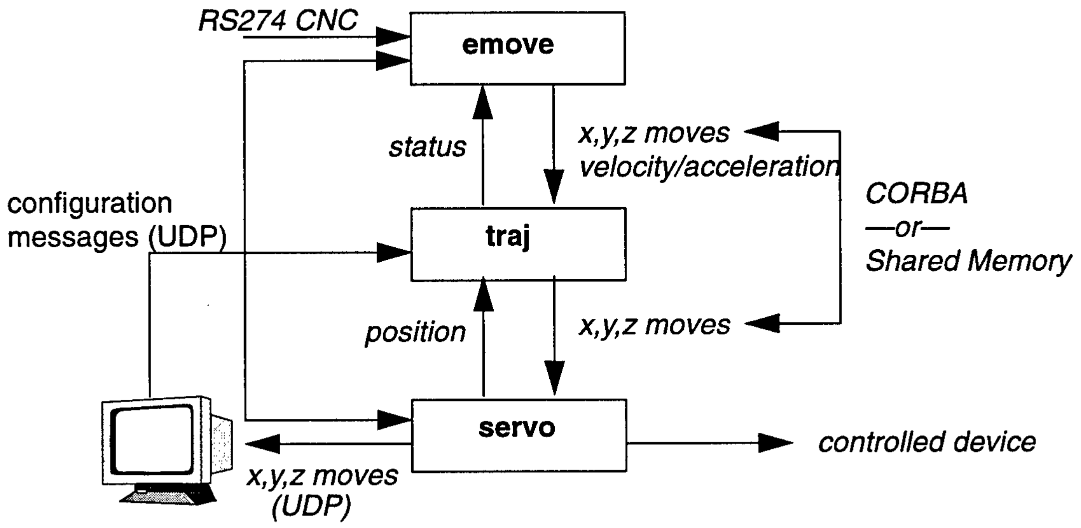


Figure 4-1: The CORBA-Based Motion Controller's Communication Structure

4.2 Observations

Figure 4-3 shows that there are some differences in the behavior of the CORBA-based motion controller demo (curves *ilu* and *ilu-mt*) compared to the shared-memory-based one (curve *shm* in Figure 4-3). The different curves in Figure 4-3 indicate that varying communication latency is the biggest problem with the CORBA-based version of the motion controller. Those variations cause `servo` to occasionally read the same trajectory point twice, which results in deceleration of the motor. Later, the `servo` component skips one or more trajectory points. As a result, the motor accelerates for a short period of time. This results in curves which are not as smooth as the curve produced by `servo` in the shared-memory-based version of the motion controller. Furthermore, variation in CORBA communication latency can cause changes to the cycle time of `traj` and `servo`. All the data shown in Figure 4-3 resulted from experiments on the LynxOS v2.4 real-time operating system.

```
INTERFACE shmServ;
(* AP 04/21/97 ILU type declarations for motion controller *)

TYPE MCell = OBJECT METHODS

     (* read/write on whole record *)
     readTrajData(OUT data: traj-struct),
     writeTrajData(IN data: traj-struct),

     (* read/write on input subrecord *)
     readTrajInput(OUT data: traj-struct),
     writeTrajInput(IN data: traj-struct),

     (* read/write on output subrecord *)
     readTrajOutput(OUT data: traj-struct),
     writeTrajOutput(IN data: traj-struct),

     (* read/write on whole record *)
     readServoData(OUT data: servo-struct),
     writeServoData(IN data: servo-struct),

     (* read/write on input subrecord *)
     readServoInput(OUT data: servo-struct),
     writeServoInput(IN data: servo-struct),

     (* read/write on output subrecord *)
     readServoOutput(OUT data: servo-struct),
     writeServoOutput(IN data: servo-struct),

END;
```

Figure 4-2: Interface Definition for `shmServ`

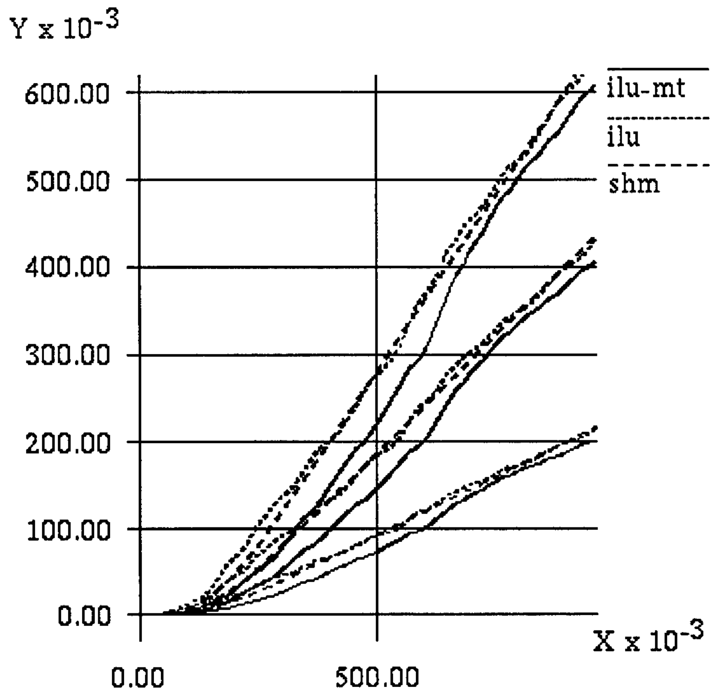


Figure 4-3: `servo` Motion to Position (x, y, z) = (1, 2, 3)

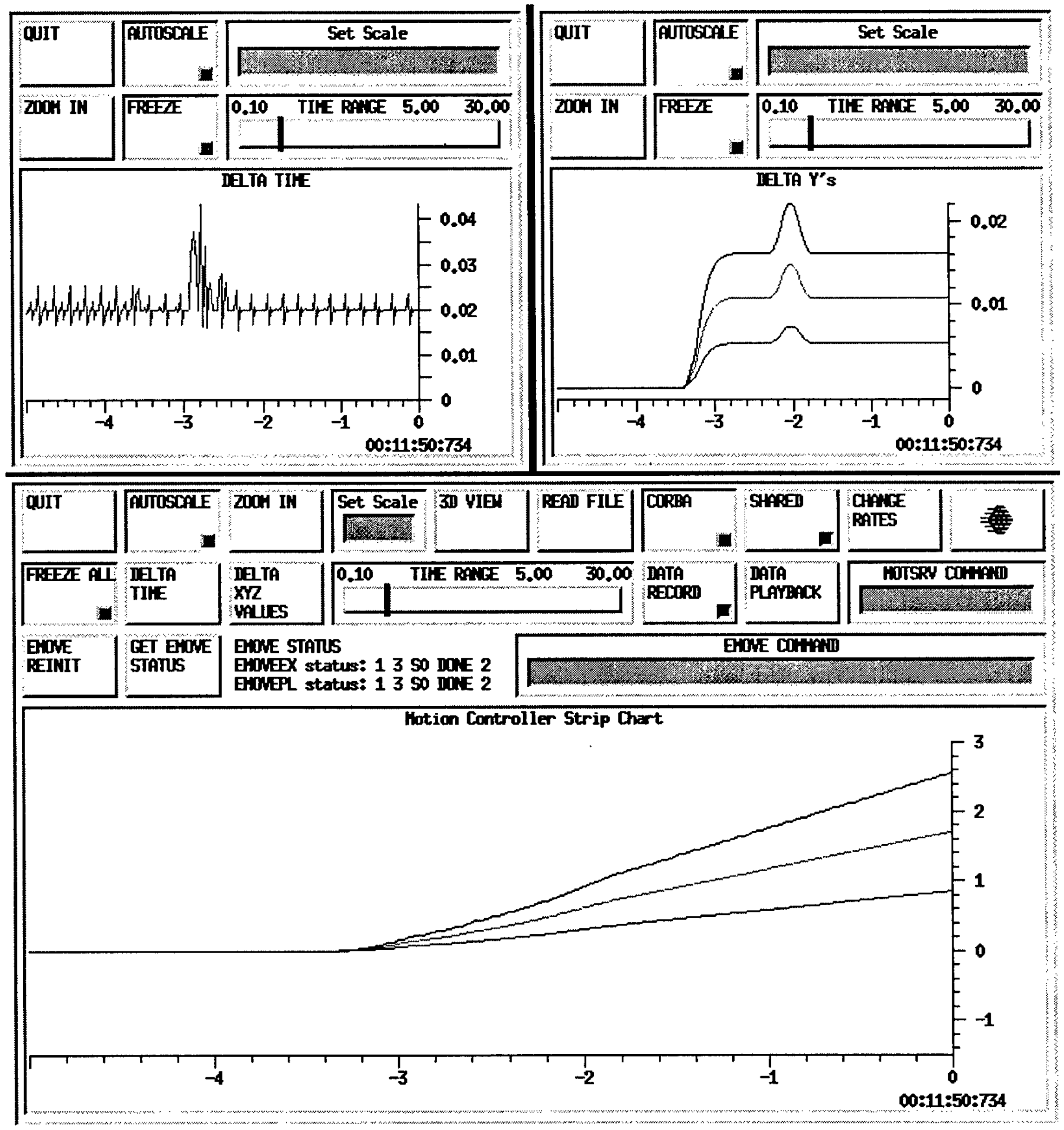
Our *initial* conclusions from the CORBA-based motion controller were not as pleasant as the ones obtained from the synchronized pendulum demo; in this scenario, CORBA seems to interfere with the real-time application. Replacement of shared-memory communication (which is implemented with predictable performance on the LynxOS) by CORBA communication significantly disturbs the motion controller's behavior.[8] We therefore conclude that simply porting an ORB (such as ILU) to the real-time platform (such as LynxOS) is not sufficient to integrate CORBA and real-time programming.

Upon closer investigation, however, we found that overall performance could be improved by "tuning" various system parameters (for example, in the RCSlib source code). Furthermore, the impact of CORBA latency may be more apparent than real; it is not clear whether the variations depicted in Figure 4-3 would have any impact on the final product being milled. We therefore conclude that a blanket assertion that today's ORBs are unsuitable for real-time systems is too strong. Instead, we need tools to help us assess the impact of variations in latency in particular settings. This is especially true in soft real-time settings where performance models are not always based upon formal scheduling theory such as RMA, but rather on heuristics

8. ILU does not provide predictable latency.

and extensive testing. As a result, performance models of many legacy soft real-time systems need to be "discovered," and it is for this purpose that tools are needed.

The SEI developed software for visualizing communication aspects of the motion controller in order to obtain visibility into the impact of CORBA latency. One set of tools is centered on a strip chart (see Figure 4-4)—a record of the x, y, and z coordinates of the `servo` plotted against time. Associated with the strip-chart view are windows that depict communication latency (the delta-time view) and `servo` acceleration (the delta y's view). The software also supports switching between shared memory and CORBA interprocess communication, capture and playback of sessions, viewing data at different scales (time and distance), and so forth.

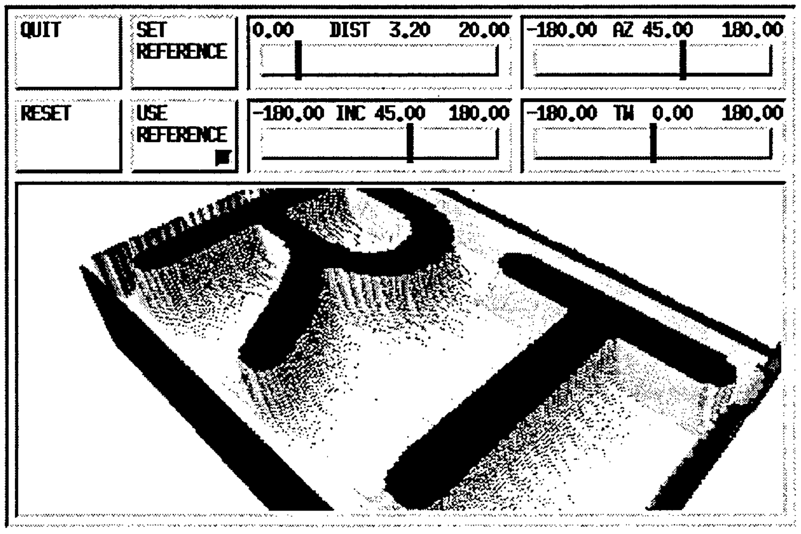


```
EMOVE STATUS
EMOVEEX status: 1 3 S0 DONE 2
EMOVEPL status: 1 3 S0 DONE 2
```

Figure 4-4: Strip Chart

A three-dimensional surface view was also developed by the SEI to obtain a more direct visualization of the impact of latency on the final (milled) product. This view is illustrated in Figure 4-5. The manufactured object can be viewed from multiple perspectives and at various distances. A *paint* feature allows the object to be painted between runs of the NC program. Thus, any paint removed by successive runs can be attributed to the effect of latency (ideally, no new paint would be removed in successive runs).

Upon more detailed investigation, our overall conclusions regarding the use of CORBA as a communication mechanism in a real-time application are (not surprisingly) mixed. Variations in communication latency do not necessarily imply negative consequences, and where there are negative consequences some of these may be manageable by performance tuning. Tools such as the strip charts depicted in Figure 4-4 and Figure 4-5 can be very useful for investigating the impact of CORBA in particular application settings. In Figure 4-5, an NC program for milling the letters "RT" from a block has been simulated using the motion controller and shared memory for interprocess communication. This tool run represents an optimal, or baseline, case, and so we use it as a "reference cut" for later comparisons. One such comparison is illustrated in Figure 4-6, where we have "painted" the reference cut and rerun the NC program, this time using the ILU ORB for interprocess communication. The paint feature, combined with the reference cut, allows us to highlight areas of both under- and over-cut (the former fails to remove material, the latter removes excess material) that result from jitter introduced by the ILU ORB.[9]



0.00 DIST 3.20 20.00
-180.00 AZ 45.00 180.00
-180.00 INC 45.00 180.00
-180.00 TW 0.00 180.00

Figure 4-5: Three-Dimensional View of Motion Controller

9. These variations are more clearly visible on a color display than in a black-and-white reproduction.

Note that the interpretation of under- and over-cuts requires that sensitivity thresholds be understood and taken into consideration. For example, we might be only interested in under-cuts that exceed .1 millimeter in depth; or, we may wish to use different color ranges to depict under-cuts of different intensities. Such features were not incorporated into the version of the visualization software depicted here. Thus, any variations in the milling surface, no matter how small, are highlighted.

While the visualization tools are useful for understanding the performance impact (both good and bad) of using CORBA in real-time settings, there is a certain flavor of *ad hoc* engineering involved in this approach, whereas guaranteed quality of service for communication services would clearly be more desirable. We propose the concept of composite objects for predictable integration of CORBA and real time. In the case of the motion controller, `emove` to `traj` communication would be handled analogously to the Simplex model problem; that is, `traj` would be split into two threads or processes. One thread would manage the CORBA-side invocations; the other would manage the real-time communication with `servo`. A central clock event would synchronize execution of those methods. All communication would be handled by a different, non-real-time thread in the composite objects. This way, the varying latency of CORBA communication could be eliminated as a source for changes to the cycle time of the periodic execution in `traj` and `servo`. Composite objects are discussed in more detail in Section 5.

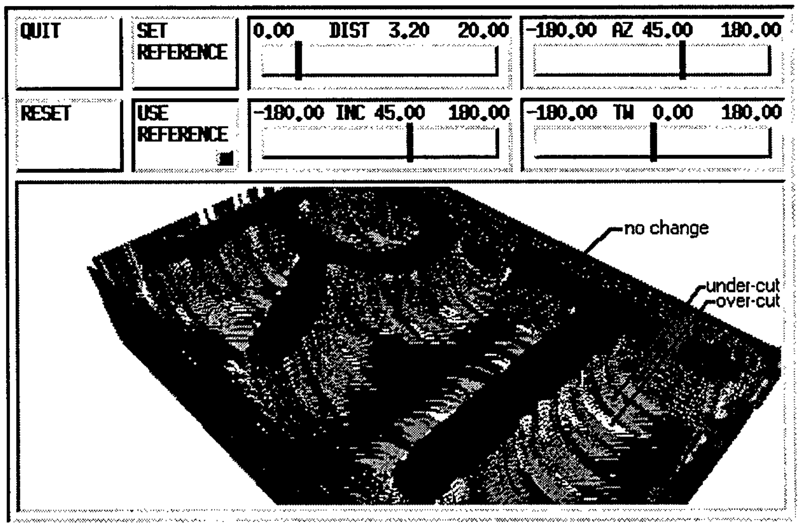


Figure 4-6: Three-Dimensional View with Under- and Over-Cut References

5 Composite Objects

Composite Objects is an approach for integrating real-time and non-real-time computing into a single object-based framework. (Figure 5-1 shows the structure of a Composite Object.) From our point of view, the real-time extension of CORBA is not a simple refinement of the existing CORBA model. Rather, it is at the heart of CORBA specifications: what should (and should not) be abstracted, and what form these abstractions should take.

We believe the goals of real-time CORBA should be to support *coexistence* and *the need to know*. Existing (by definition non-real-time) ORBs should be able to share resources and interoperate with real-time applications without introducing destructive interference (i.e., coexistence). Further, only clients requiring real-time quality of service should be required to deal with the extra level of complexity implied by such guarantees; on the other hand, these necessary complexities should be exposed to applications that do have real-time requirements (i.e., need to know).

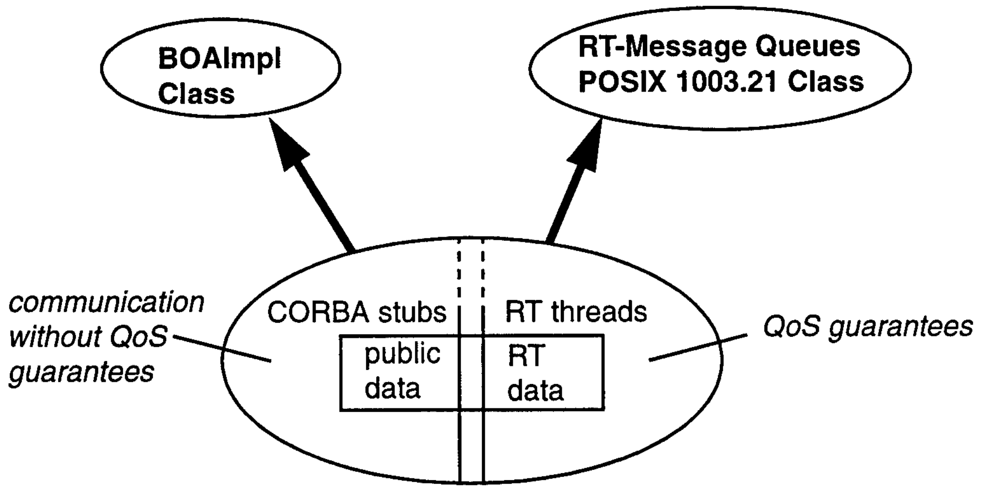
There are three design rules underlying composite objects that we believe will help achieve our goals of *coexistence* and *need to know*:

1. non-interference: We should create an environment in which general purpose computing and real-time computing will not burden each other.
2. interoperability: The services exported by general purpose computing objects and real-time computing objects can be used by each other.
3. adaptive abstraction: Lower level information and scheduling actions needed by real-time computing is available for real-time objects but transparent to non-real-time objects.

Rule 1 is implemented by partitioning the system's computation and communication resources. Such partitions can be done physically or logically. Rules 2 and 3 will be implemented by composite objects.

Composite objects provide a unified interface for interactions between real-time and non-real-time objects. This interface will be identical to existing CORBA definitions from the viewpoint of non-real-time objects. Non-real-time clients can invoke the service of real-time objects via the standard CORBA interface. It is the task of the composite object to ensure that the service to non-real-time clients will be carried out in best effort without adversely affecting the real-time computation. This way, non-real-time components like graphical user interfaces and databases can be connected to existing real-time systems.

Real-time clients can obtain services from standard (non-real-time) CORBA objects. In that case, the composite object will act as the surrogate of the real-time object in getting the service and forwarding it to the real-time object, so that real-time clients will not wait for the unpredictable timing of non-real-time services. An important application of this service is to identify potential real-time servers registered with CORBA. Once real-time clients and servers are identified and their resources are reserved, execution of their methods and creation of data flow paths can be managed by sending commands to Simplex architecture servers.


BOAImpl Class
RT-Message Queues POSIX 1003.21 Class
communication without QoS guarantees
CORBA stubs
RT threads
public data
RT data
QoS guarantees

Problems:
- **scheduling of RT threads**
- **firewall/mirroring between RT & public data**

Figure 5-1: Structure of a Composite Object

Since composite objects provide functionality to respond to CORBA-method invocations, they can be seen as descendants of a class which implements CORBA object adaptor functionality (e.g., the CORBA-defined Basic Object Adaptor). On the other hand, composite objects also need the capability to create real-time programming abstractions like prioritized threads and real-time communication channels. Thus, they can be seen as descendants of a class which implements real-time servers such as Simplex architecture servers.

Figure 5-1 shows the overall structure of a composite object. Composite objects consist of a real-time part and a non-real-time part. Design time and runtime guarantees can be given for execution of *time-triggered* methods and *real-time service* methods, respectively. In contrast, methods in the non-real-time part of the composite object are executed using a best-effort approach. Those non-real-time methods correspond to the well-known, standard, CORBA-method invocations. The principle of adaptive abstraction is realized here by hiding all real-time implementation details from the CORBA user, whereas scheduling and timing information is accessible for the real-time part of a composite object.

A composite object's real-time services include

- time-triggered methods: These methods encapsulate time-triggered actions such as the periodic sampling of a sensor device.

- real-time service methods: These methods encapsulate data driven service at a given level of priority and with a worst case execution time (e.g., a particular filtering method in a filter object).
- data flow service: These methods encapsulate real-time communication services such as the real-time publish and subscribe [Rajkumar 95].

The invocation of these methods can be either accepted or rejected, subject to the decision of the schedulability analysis, for example, as provided by Simplex replacement transaction servers. If all the requested client objects, server objects, and their data flow paths are schedulable, they will be created, and a new real-time service will be created at runtime. Otherwise, the request will be rejected. Thus, composite objects establish timing firewalls [Polze 97] between real-time and non-real-time (CORBA) computing, so that the non-real-time part cannot violate the real-time scheduling rules [Sha 94] that are needed by the real-time part.

Currently, composite objects is an untested design concept. Although some aspects of the inverted pendulum model problem reflect composite object techniques (e.g., thread priority separation), other aspects of composite objects remain to be demonstrated (e.g., call admission). Future work will involve the development of a prototypical implementation of the composite objects. For example, we assume that the composite objects will allow us to solve the motion controller's problems with periodic execution which are apparently introduced by varying communication latency of CORBA.

6 Related Work

So far, relatively little work concerning the integration of CORBA and real-time programming techniques and environments has been published.

The Object Management Group (OMG) has established a CORBA real-time special interest group (RTSIG). A white paper on real-time CORBA was released on November 15, 1996 [McGoogan 96]. Other activities include the development of requests for proposals (RFPs) that would lead to ORBs that satisfy the needs of real-time computing applications. In particular, the RFPs will address time services, fixed priority scheduling, and a "minimal ORB" (small footprint). However, the future direction of the RTSIG is a matter of speculation.

ANSA (Advanced Networked Systems Architecture) [ANSA] is an open, collaborative research program managed by the British company APM Limited. The initial programming interface supported by ANSA follows Open Distributed Processing (ODP) standards. However, the current project, called *Jet,* was started to provide a CORBA application programming interface (API) to the previously released ANSA/ODP API. Jet provides its own CORBA IDL compiler and a subset of the CORBA 2.0 C++ mapping. The project's objective is to extend CORBA with real-time multimedia functionality, such as streams, signals, explicit binding, and quality of service (QoS). Jet seeks to provide interoperability with CORBA platforms; it should be possible to run management applications to control real-time applications from remote CORBA platforms.

Work at the University of Rhode Island and the MITRE Corporation deals with syntactical extensions to CORBA IDL to express timing constraints [Wolfe 95, Thuraisingham 96]. A proposed implementation uses four CORBA context declarations (`_after`, `_before`, `_by`, `_execute`), to specify deadlines for transmission of data between client and server and for execution of the server's method. *Timed distributed method invocations* are identified as one necessary feature in a real-time distributed computing environment. A *Global Time Service*, *Real-Time Scheduling of Services*, a *Global Priority Service*, and *Bounded Message Latency* are identified as prerequisites for the proposed approach.

TAO, a new "end-system architecture" for CORBA-based systems [Gokhale 97], is designed to provide end-to-end QoS guarantees for CORBA applications. A list of requirements for ORB implementations is presented; among these requirements are resource reservation protocols, optimized real-time communication protocols, and a real-time object adapter. However, in contrast to the composite objects idea of interfacing an existing real-time system with CORBA, the TAO approach focuses on completely new, CORBA-based, real-time systems.

7 Conclusions

CORBA is a popular and important technology, and it is not surprising that there is interest in using CORBA in real-time application settings. Unfortunately, CORBA was not designed for real time, and, moreover, real-time computing remains a niche (albeit an important niche) in the overall marketplace. Since the OMG is attempting to respond to this overall marketplace (as are the majority of existing ORB vendors), there is reason to be skeptical that CORBA will evolve to address the full range of real-time issues (priority scheduling, real-time clocks, etc.).

On the other hand, this does *not* invalidate the use of CORBA in *all* real-time application settings. In this paper, we have reported on our experiences in using CORBA to provide implementations to two model problems, each of which we believe to be representative of a larger class of applications in the manufacturing domain. In one model problem, we explored the use of CORBA as a non-real-time gateway between two real-time (and fault-tolerant) applications. In a second model problem, we explored the use of CORBA as a communication mechanism within a real-time system. In both model problems, our theme was the use of available ORBs (as opposed to hypothetical, or prototypical extensions to CORBA). A secondary theme was the integration of existing real-time applications with ORBs.

Based upon our experience, ORBs available today can be used with a reasonable degree of confidence as a non-real-time gateway between real-time subsystems. Provided that remote object executions do not disturb the real-time assumptions of the integrated applications (a large proviso), the communication latency and variations in latency of existing ORBs should be suitable for many applications. The Simplex model problem illustrated this point via a non-real-time, graphical, human-machine interface—where latency factors would have an obvious and visible impact. On the other hand, the proviso of non-interference is not guaranteed easily, and we were able to construct pathological scenarios to illustrate the interference of CORBA with real-time assumptions.

Using CORBA as a communication mechanism within a real-time system introduces far greater demands upon an ORB. First, communication latency may become far more consequential as deadlines may depend upon *predictable* latency; CORBA provides no direct support for any such quality-of-service guarantee. Second, in these scenarios the ORB takes on the role of shared resource (CORBA communication is accomplished via method execution on *server objects*), thus introducing the specter of priority inversion. Without a priority-based scheduling policy, an ORB will be hard pressed to avoid such problems. However, variation in latency is not an *a priori* justification to reject CORBA in a real-time setting; we need to be able to quantify the impact of jitter (variation of latency) on the actual manufacturing process. Similarly, there may be suitable design "workarounds" if it is known that an ORB is susceptible to priority inversion. In short, to be forewarned is to be forearmed, and what is important is an awareness of both the limitations of an ORB as well as the impact of these limitations on the actual application. The strip-chart software demonstrates how visibility into the impact of ORB-induced jitter can be obtained.

Composite objects allow us to generalize our approaches for integrating non-real-time ORBs with real-time applications. We can view composite objects as a repair strategy for removing (or resolving) mismatched assumptions between ORBs and real-time applications. The overall goal of composite objects is to allow reliable integration of available off-the-shelf ORBs with real-time systems, rather than requiring (or depending on) the success of the OMG in promulgating a new and improved CORBA specification. The design principles of composite objects are non-interference, interoperability, and adaptive abstraction. In this paper we outlined several ideas for developing composite objects, but to date these ideas remain untested. Continuing work at the SEI and Humboldt University of Berlin will be focused on developing prototypical implementations of composite objects.

References

[ANSA] Advanced Networked Systems Architecture. Multiple articles available on the WWW.<URL: http://www.ansa.co.uk/ANSA/-index.html>.

[Brown 96] Brown, A. & Wallnau, K. "A Framework for Evaluating Software Technology." *IEEE Software 14, 2* (September 1996): 39-49.

[Gokhale 97] Gokhale, A.; Schmidt, D. C.; Harrison, T. H.; & Parulkar, G. "A High-Performance Endsystem Architecture for Real-Time CORBA." *IEEE Communications 14*, 2 (February 1997): 72-77.

[McGoogan 96] McGoogan, J., ed. "Realtime CORBA - A White Paper - Issue 1.0." *OMG Real-Time Platform SIG*, Initial Review Draft. November, 1996.

[NIST] National Institue of Standards and Technology. "Intelligent Controller Systems Development." Articles available on the WWW . <URL: http://isd.cme.nist.gov/brochure/SoftwareSystems.html>.

[OMG 95] Object Management Group. *The Common Object Request Broker.* Framingham, MA: Object Management Group, Inc., 1995.

[Pollak 96] Pollak, B., ed. *Portable Operating System Interface (POSIX) - Part 1x: Real-Time Distributed Systems Communication Application Program Interface (API).* IEEE Standard, P1003.21 LIS/V1.0, September 1996.

[Polze 97] Polze, A.; Fohler, G.; & Werner, M. "Predictable Network Computing," 423-431. *Proceedings of the International Conference on Distributed Computing Systems (ICDCS'97).* Baltimore, MD, May, 1997. New York, NY: IEEE Computer Society Press.

[Rajkumar 95]	Rajkumar, R.; Gagliardi, M.; & Sha, L."The Real-Time Publisher/Subscriber Interprocess Communication Model for Distributed Real-Time Systems: Design and Implementation," 66-75. *Proceedings of the First IEEE Real-Time Technology and Applications Symposium*. Chicago, IL, May 15-17, 1995. New York, NY, IEEE Computer Society Press, 1995.
[Schmidt 97]	Schmidt, D. C.; Gokhale, A.; Harrison, T. H.; Levine, D.; & Cleeland, C. "TAO: a High-Performance Endsystem Architecture for Real-Time CORBA." RFI response to the OMG Special Interest Group on Real-time CORBA, 1997.
[Sha 94]	Sha, L.; Rajkumar, R.; & Sathaye, S. S. "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems." *Proceedings of the IEEE 82*,1 (January 1994): 68-82.
[Sha 96]	Sha, L.; Rajkumar, R.; & Gagliardi, M. "Evolving Dependable Real-Time Systems," 335-346. *Proceedings of the 1996 IEEE Aerospace Applications Conference*. Aspen, CO, February 1996. New York, NY: IEEE Computer Society Press, 1996.
[Soley 95]	Soley, R. M., ed. *Object Management Architecture Guide*, 3rd ed. New York, NY: John Wiley & Sons, 1995.
[Thuraisingham 96]	Thuraisingham, B.; Krupp, P.; & Wolfe, V. "Position Paper: On Real-Time Extensions to Object Request Brokers," 182-185. *Proceedings of the Second Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*. Laguna Beach, CA, February 1996. New York, NY: IEEE Computer Society Press, 1996.
[Wolfe 95]	Wolfe, V. F.; Black, J. K.; Thuraisingham,B.; & Krupp, P. "On Real-Time Extensions to Object Request Brokers: A Panel Position Paper," 182-185. *Proceedings of the Second Workshop on Object-Oriented Real-Time Dependable Systems*. Los Alamitos, CA, February 1996. New York, NY: IEEE Computer Society Press, 1996.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.			
1. AGENCY USE ONLY (leave blank)	2. REPORT DATE December 1997		3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE A Study in the Use of CORBA in Real-Time Settings: Model Problems for the Manufacturing Domain			5. FUNDING NUMBERS C — F19628-95-C-0003
6. AUTHOR(S) Andreas Polze, Daniel Plakosh, Kurt C. Wallnau			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213			8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-97-TR-011
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/AXS 5 Eglin Street Hanscom AFB, MA 01731-2116			10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-97-011
11. SUPPLEMENTARY NOTES			
12.a DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS			12.b DISTRIBUTION CODE
13. ABSTRACT (maximum 200 words)			
14. SUBJECT TERMS Common Object Request Broker Architecture (CORBA), composite objects, manufacturing, object request brokers (ORBs), model problems, real-time requirements			15. NUMBER OF PAGES 40
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is an important and popular technology that supports the development of object-based, distributed applications. The benefits promised by CORBA (abstraction, heterogeneity, etc.) are appealing in many application domains, including those that satisfy real-time requirements—such as manufacturing. Unfortunately, CORBA was not specified in light of real-time requirements, and so the question remains whether existing object request brokers (ORBs) can be used in real-time settings, or whether developers of real-time systems must await future extensions of CORBA that address real-time issues or use non-CORBA- compliant ORBs. In this report, we describe the application of an off-the-shelf ORB to two real-time model problems. Based on our experiences, we believe that today's ORBs can be used in real-time settings, with certain caveats as outlined in this report. We also outline the concept of *composite objects*, an approach for extending the range of non-real-time ORBs into a greater variety of real-time settings.